Predicting Changes in Exchange Rates and Comparing Predictions of Different Machine Learning Methods

Aalto University, August 2022.

I. INTRODUCTION:

Currency fluctuations are a natural outcome of floating exchange rates, which is the norm for most major economies. Various factors influence exchange rates, including a country's economic performance, the outlook for inflation, interest rate differentials, capital flows and so on.

A currency's exchange rate is typically determined by the strength or weakness of the underlying economy. As such, a currency's value can fluctuate from one moment to the next.

The main goal of this project is to help brokers and businesses make better decisions in crisis situations and to help governments to control cash flow and international transactions, and avoid crime like money laundering.

In order to forecast these critical situations, we could predict the future exchange rates of the currency based on the current and past rates, by using machine learning. We would train some models on a dataset we found on the European Central Bank and try to predict the future Swedish Krona rates.
From the previous dataset, we will explicitly use exchange rates from a specific day and use them as features, and the SEK-EUR exchange rate will be our main label.

Then we will apply two machine learning models on the training set and predict the results using the validation set. We will compare both models results and performances using loss functions, and determine the most efficient model.

II. PROBLEM FORMULATION

The problem consists of three important concepts i.e., data points, models and loss functions. Data points are some objects that carry relevant information[1], which are the exchange rates from a specific day in the project. The data points are collected from 4:th January 1999 to eighth August 2022, which are 6048 data points. Data points can also be presented by features and labels. Features are exchange rates from EUR to DKK, EUR to USD and EUR to NOK and our label is the exchange rate from EUR to SEK.

Models are hypothesis spaces and the loss functions are quality measures[2]. The main purpose is to fit models to data that consists of a set of data points, in order to get accurate predictions of the exchange rate in the future. Thereafter, the loss functions are used to measure errors' sizes and qualities of the models.

Loss functions that are applied in the project are Mean Squared Error Loss(MSE) and Huber Error Loss. MSE is the sum of squared errors, which means that it is very sensitive to outliers because the penalty is squared. But the MSE has a form of quadratic equation as shown in Equation 1[3] and Figure 1, 2, which

[1]Alexander Jung, "Three Components of Machine Learning ", lecture notes, Department of Computer Science, Aalto university, Helsinki, Finland, August 2022.

[2] Alexander Jung, "Three Components of Machine Learning ", lecture notes, Department of Computer Science, Aalto university, Helsinki, Finland, August 2022.

[3] https://www.numpyninja.com/amp/loss-functions-when-to-use-which-one

ensures that there is a gradient descent with only one global minimum value.

As seen in equation 2[4], Huber Error Loss is a combination of MSE and MAE that is the average of absolute errors. As shown in figure 3 and 4, if the prediction error is too big, the penalty becomes MAE, which decreases the sensitivity of outliers.

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{predicted})^2$$

Equation 1, formula for MSE.

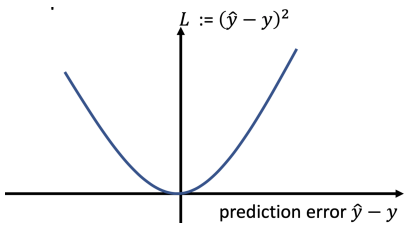


Figure 1, illustration of MSE.

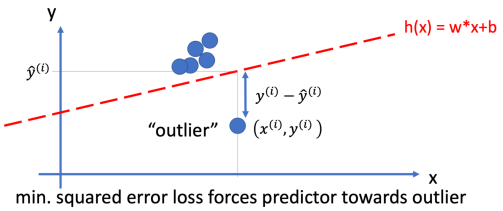


Figure 2, illustration of outliers for MSE.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Equation 2, formula for Huber Error Loss

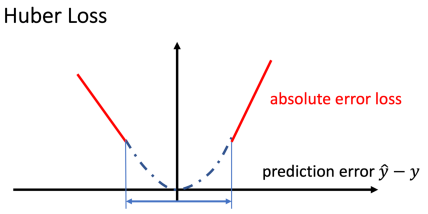


Figure 3, illustration of Huber Error Loss

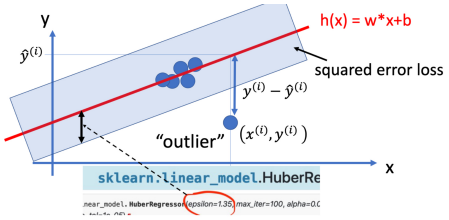


Figure 4, illustration of outliers for Huber Error Loss.

As a benchmark we use the last two years' worth of data to validate the performance of the model in a very unpredictable economical situation.

III. METHODS

For Predicting the exchange rates from EUR to SEK we used two different kinds of models. The first one is the Multiple Linear Regression model. It uses linear maps to predict the label for a given feature vector. We chose this model, because the multiple variables that are used in this model helps us to be better armed against the fluctuations of the Exchange Rate. For the implementation of the Linear Regression we use the sklearn.linear_model.LinearRegression class.

The Linear Regression uses the Mean Squared Error (MSE) as a Loss function, which makes it prone to outliers, but because of the quadratic formula it ensures that it has only global minimum. For calculating the MSE we used the

4 https://www.numpyninja.com/amp/loss-functions-when-to-use-which-one

sklearn.metrics.mean_squared_error method.
Our second model is the Huber Regression. It also uses linear maps to predict labels. We chose it because it uses a different loss function than the Linear Regression, which makes it more robust to outliers. For loading and training this model we used the sklearn.linear_model.HuberRegressor class.

The Huber Regression uses the huber loss as a Loss function. In a bandwidth, which is determined by the value of epsilon, it uses the MSE and outside of this bandwidth it uses the absolute error. The combination of these methods makes the huber loss more robust against outliers. To calculate the Huber loss we used our own implementation, which can be seen in the notebook which is added at the end.

We got our data from the European Central Bank (ECB). The data is publicly available at https://sdw.ecb.europa.eu/. We splitted in a training (1999-01 to 2019-12), validation (2020-01 to 2021-12) and test set (2022-01 to 2022-08).

We didn't use a Polynomial Model, because of the higher risk of overfitting.

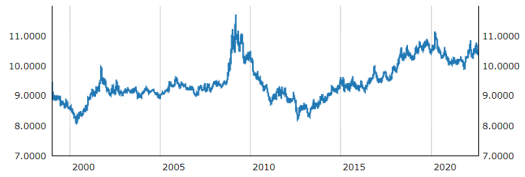


Figure 5: Development of the exchange rate from EUR to SEK in the period from 1999 to 2022[5]

5 https://sdw.ecb.europa.eu/quickview.do;jsessionid=0E0F065884D3C58E6692DA77BF52E2CF?SERIES_KEY=120.EXR.D.SEK.EUR.SP00.A&resetBtn=+Reset+Settings&start=&end=&trans=N

IV. RESULTS

The appended notebook provides the used implementation of the methods described in the previous section, and they return the following errors:

Method	Training error	Validation error
Linear	0.1692	0.1473
Huber	0.0874	0.0228

Table 1: Training and Validation errors

From this table it can be seen that Huber Regression returns approximately half of the training and validation errors when compared to Linear Regression. This would imply that Huber regression is better for this data, but there might still be overfitting or a prediction that doesn't make sense. Because of this, the results are also compared visually by plotting them both. Due to the nature of the data and prediction, it is illustrated by plotting them as time series with the predictions appended to the end, on top of the true labels.

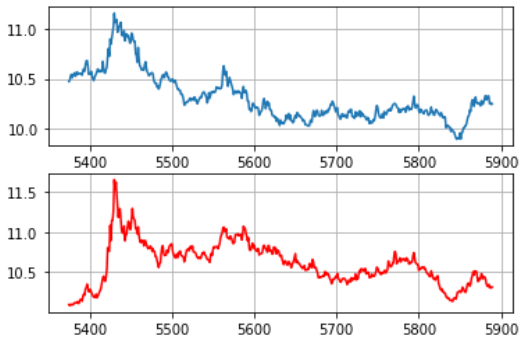


Figure 6: True labels (blue) and Linear Regression prediction (red) for the Validation set

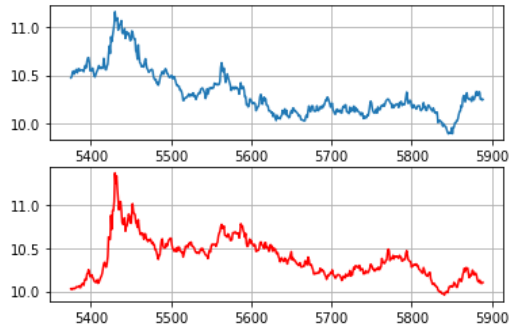


Figure 7: True labels (blue) and Huber Regression prediction (red) for the Validation set

It can be observed from Figures 6 and 7 that the predictions are very nearly equivalent. However, as can be seen in the results in Table 1, Huber Regression appears to result in more accurate predictions on average regardless.

Analyzing the residuals of a model is another method of evaluating its performance. Residuals are assumed to be i.i.d. and follow a normal distribution with a mean of 0. Thus, the residuals of a well-fitted model follow such a distribution.

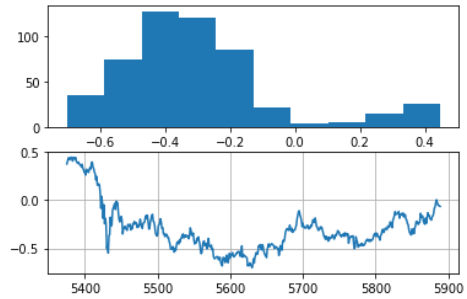


Figure 8: Linear Regression residuals as histogram and time series plots for the Validation set

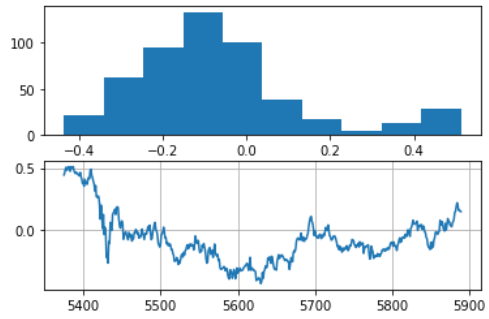


Figure 9: Huber Regression residuals as histogram and time series plots for the Validation set

A statistical test is not needed to conclude that the residuals of either model are not normally distributed, which is not ideal. Both Figures 8 and 9 show signs of being skewed to the right.

Based on the chosen error measurements, the visual evaluation of predictions and the residual analysis, Huber loss was chosen for Test evaluation. The calculated test error of the model for the test set is 0.0951.

V. CONCLUSIONS

In this report we forecast exchange rates from EUR to SEK by using Multiple Linear Regression and Huber Regression models. Table 1 shows us that both models perform better on the validation set than on the training set, which shows that there is not much overfitting. Furthermore the validation error of the Huber Regression is much lower than the error of the linear regression model, which shows that Huber Regression is the better choice for predicting exchange rates. However if we look at the prediction error of unseen data the Linear Regression model (0.07) performs better than the Huber Regression model (0.095). This difference is critical when investing high amounts of money in exchange rates. So in this case we advise using Linear Regression instead of Huber Regression.

In this report we used Machine Learning methods to predict Time Series Data. The methods performed well, considering unforeseeable events like the pandemic. Further Research is needed to see if ML methods could challenge usual times series methods for forecasting this kind of data.

VI. REFERENCES

Alexander Jung, "Three Components of Machine Learning ", lecture notes, Department of Computer Science, Aalto university, Helsinki, Finland, August 2022.

https://www.numpyninja.com/amp/loss-functions-when-to-use-which-one

https://sdw.ecb.europa.eu/quickview.do;jsessionid=0E0F065884D3C58E6692DA77BF52E2CF?SERIES_KEY=120.EXR.D.SEK.EUR.SP00.A&resetBtn=+Reset+Settings&start=&end=&trans=N

VII. APPENDIX

PredictingExchangeRates

August 27, 2022

[ ]:
```
from google.colab import drive

drive.mount("/content/gdrive")
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

Importing Libraries

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import (LinearRegression, HuberRegressor)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
```

Implementing the Huber Loss

[ ]:
```
def HuberLoss(error, epsilon):
    #error - prediction error
    #epsilon - within the space of epsilon the MSE is used
    error = abs(error)
    bigger_than_epsilon = (error > epsilon)
    return (~bigger_than_epsilon) * (0.5 * error ** 2) - (bigger_than_epsilon)␣
 ↪* epsilon * (0.5 * epsilon - error)
```

Loading and Splitting the Data

[ ]:
```
#load data
df = pd.read_excel('/content/gdrive/MyDrive/Exchange_Rates_Dataset.xlsx')␣
 ↪#(DON'T REMOVE)
#df = pd.read_excel('/content/sample_data/Exchange_Rates_Dataset.xlsx')
#df = pd.read_excel('/content/gdrive/My Drive/Exchange_Rates_Dataset.xlsx')
days_from_start = (df.index -  df.index[0]).to_numpy()
df["days from start"] = days_from_start
display(pd.DataFrame(df))
data = df.to_numpy()
data = np.delete(data, 0, 1)
```

```
#split data
#5375th row corresponds to 2019-12-31
#5890th row corresponds to 2021-12-31
y_train = data[:5375, 1] #training labels -> Swedish Krona from 1999-01-04 to␣
 ↪2019-12-31
y_val = data[5375:5890, 1] #validation labels -> Swedish Krona from 2020-01-02␣
 ↪to 2021-12-31
y_test = data[5890:,1] #test labels -> Swedish Krona from 2022-01-03 to␣
 ↪2022-08-11
y_tot = data[:,1]
data = np.delete(data, 1, 1)
X_train = data[:5375] #training features -> (Danish, Norwegian Krona, US␣
 ↪Dollar, Date) from 1999-01-04 to 2019-12-31
X_val = data[5375:5890] #validation features -> (Danish, Norwegian Krona, US␣
 ↪Dollar, Date) from 2020-01-02 to 2021-12-31
X_test = data[5890:] #test features -> (Danish, Norwegian Krona, US Dollar,␣
 ↪Date) from 2022-01-03 to 2022-08-11
#display(data)

tot_time = data[:, 3]
train_time = data[:5375, 3]
val_time = data[5375:5890, 3]
test_time = data[5890:, 3]
```

	Period\Unit:	[Danish krone]	[Swedish krona]	[Norwegian krone]	\
0	1999-01-04	7.4501	9.4696	8.8550	
1	1999-01-05	7.4495	9.4025	8.7745	
2	1999-01-06	7.4452	9.3050	8.7335	
3	1999-01-07	7.4431	9.1800	8.6295	
4	1999-01-08	7.4433	9.1650	8.5900	
...	...	...	...	...	
6042	2022-08-05	7.4415	10.3573	9.9820	
6043	2022-08-08	7.4405	10.3650	9.9405	
6044	2022-08-09	7.4407	10.3875	9.9365	
6045	2022-08-10	7.4397	10.3773	9.9118	
6046	2022-08-11	7.4395	10.3600	9.8040	

	[US dollar]	days from start
0	1.1789	0
1	1.1790	1
2	1.1743	2
3	1.1632	3
4	1.1659	4
...	...	...
6042	1.0233	6042

```
6043        1.0199              6043
6044        1.0234              6044
6045        1.0252              6045
6046        1.0338              6046

[6047 rows x 6 columns]
```

1 Linear Regression

Loading and Training the Linear Model

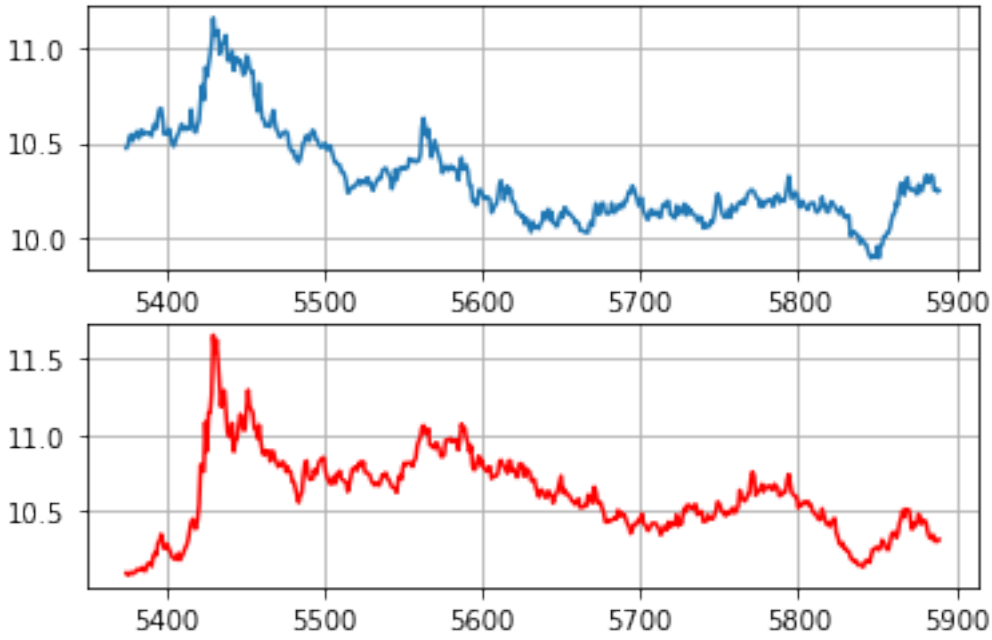
[ ]:
```
reg = LinearRegression(fit_intercept=True)
reg.fit(X_train,y_train)
linear_score = reg.score(X_train, y_train)

#printing the coefficients and the intercept of the linear regression fucntion
print("Model Coefficient of Determination:", linear_score)
print("Coefficients of Linear Regression Model :", reg.coef_)
print("Intercept of Linear Regression Model: ", reg.intercept_)

#NOTE: I would like to use a DataFrame with the features names and the␣
 ↪coefficient, cant do it rn.
```

```
Model Coefficient of Determination: 0.5167720942299134
Coefficients of Linear Regression Model : [-3.33425203e+00  6.45260146e-01
9.32138195e-01 -1.55385242e-05]
Intercept of Linear Regression Model:  27.69977926706821
```

Prediction

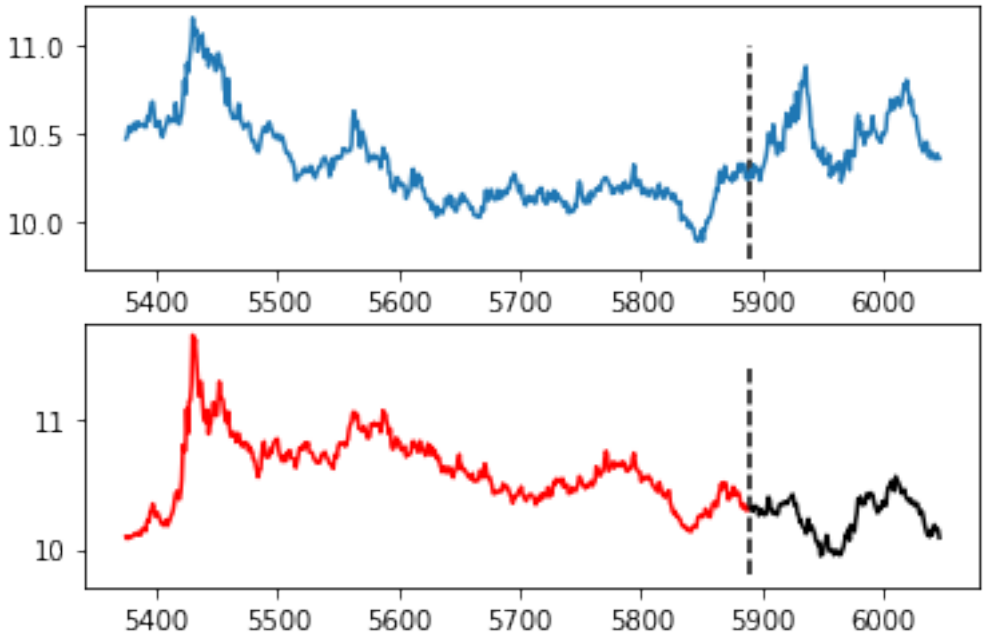
[ ]:
```
# Predicting using the model
y_pred_train = reg.predict(X_train) #prediction based on Training data
y_pred_val = reg.predict(X_val) #prediction based on Validation data
y_pred_test = reg.predict(X_test) #prediction based on Test data

#plt.scatter(y_test, y_pred_test)

#NOTE by SUKEYNA: if you see the plot the predictions and the testing values␣
 ↪are really quite far from each other....

#NOTE by Patrick: I think a scatter plot doesn't really suit this data, as we␣
 ↪are dealing with a timeseries.
#                  In this case a time-y_train plot would be better, with y_val␣
 ↪and y_pred_val appended to the end. In this way it is easy to see if it is a␣
 ↪realistic prediction.
```

```
# PLOTTING THE TRUE LABEL & PREDICTION
plt.subplot(2,1,1)
plt.plot(tot_time[5375:5890], y_tot[5375:5890])
plt.grid()
plt.subplot(2,1,2)
plt.plot(val_time, y_pred_val, 'r')
plt.grid()
```

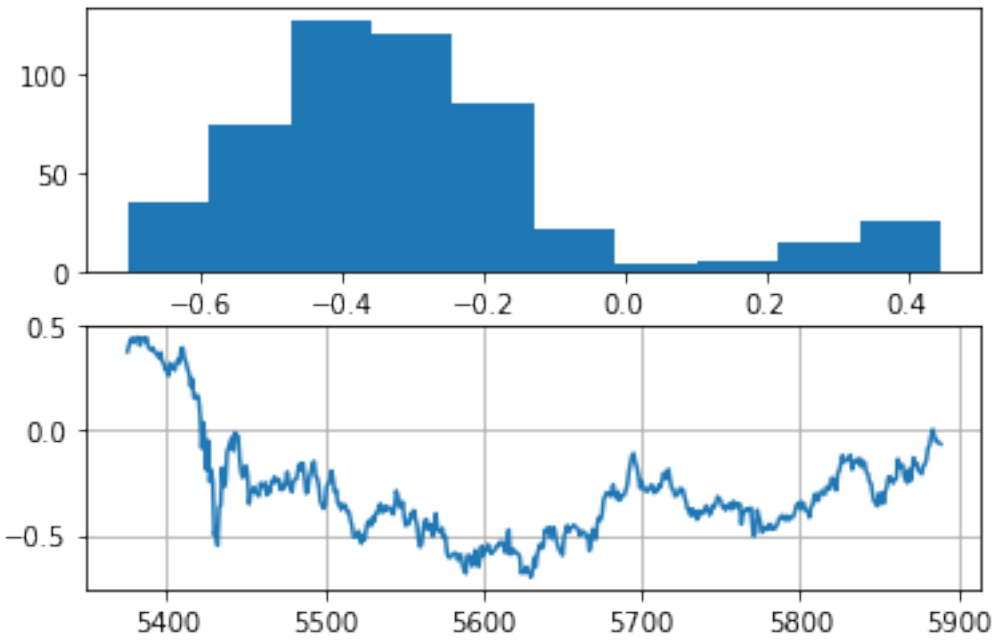


[ ]:
```
# PLOTTING THE TRUE LABEL & PREDICTION
plt.subplot(2,1,1)
plt.plot(tot_time[5375:], y_tot[5375:])
plt.vlines(5890,9.8,11, linestyles='dashed')
plt.subplot(2,1,2)
plt.plot(val_time, y_pred_val, 'r')
plt.plot(test_time, y_pred_test, 'k')
plt.vlines(5890,9.8,11.4, linestyles='dashed')
```

[ ]:
```
<matplotlib.collections.LineCollection at 0x7fb5d5d42c10>
```



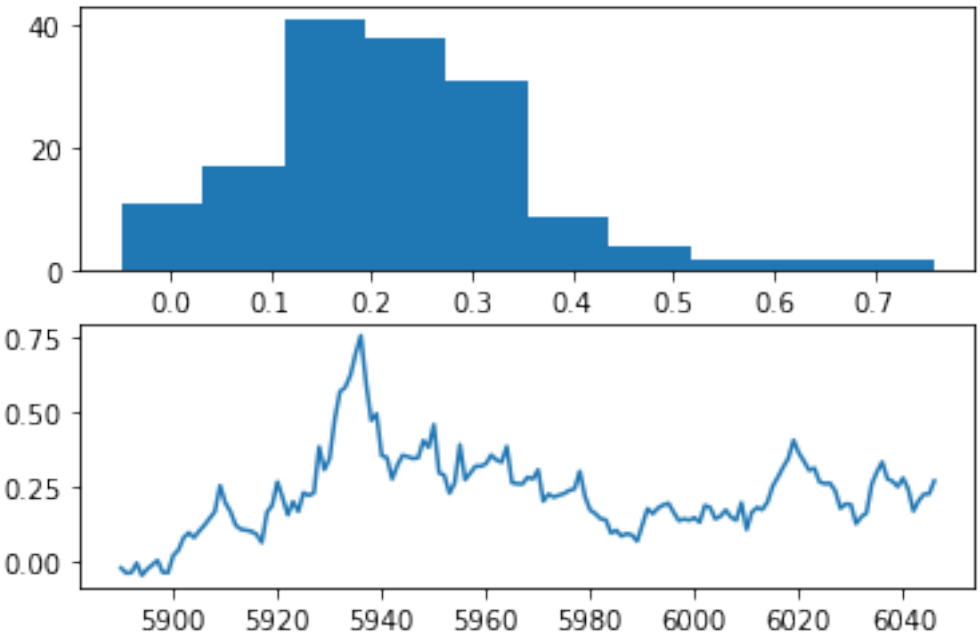
[ ]:
```
# Plotting the VALIDATION Residuals
plt.subplot(2,1,1)
val_residuals = y_val - y_pred_val
plt.hist(val_residuals)
#display(np.transpose(y_hub_pred_test))
plt.subplot(2,1,2)
plt.plot(val_time, val_residuals)
plt.grid()
```



Plotting the Residuals and Calculating the Training and Validation Loss

[ ]:
```
#Plotting the TEST Residuals (Histogram + Timeseries)
plt.subplot(2,1,1)
plt.hist(y_test - y_pred_test)
test_residuals = y_test - y_pred_test
#display(np.transpose(y_hub_pred_test))
plt.subplot(2,1,2)
plt.plot(test_time, test_residuals)

# OLD NOTES WITH OLD DATA AND OLD SCATTERPLOTS
#NOTE BY SUKEYNA: the histogram shows no normality due to many outliers :/ idk␣
 ↪if it will be an issue, hope not
#NOTE by Patrick: probably a result of the two distinct clusters that can be␣
 ↪seen in the prediction scatterplots for both implementations
#                  as we can see two normal-like "humps".

#calculate Training and Validation Loss
err_train = metrics.mean_squared_error(y_train ,y_pred_train) #training error
err_val = metrics.mean_squared_error(y_val, y_pred_val) #validation error
err_test = metrics.mean_squared_error(y_test, y_pred_test) #test error

# Printing the errors
print("Training Error: ", err_train)
print("Validation Error: ", err_val)
print("Test Error: ", err_test)
```

```
Training Error:  0.16918461336876195
Validation Error:  0.147306673530132
Test Error:  0.07088783548686102
```



2 Huber Regression

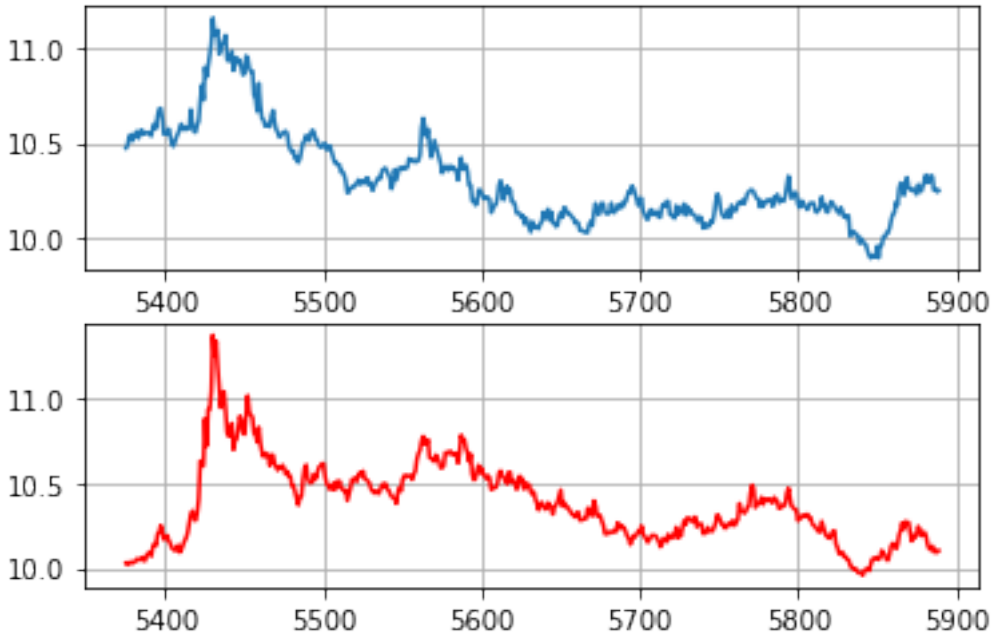
Loading and Training the Model

[ ]:
```
y_train = y_train.ravel()
huber = HuberRegressor(fit_intercept=True, epsilon=1).fit(X_train, y_train)␣
 ↪####Important Change: Epsilon has changed from 0.2 to 1, because sklearn␣
 ↪defines it to be greater or equal to 1#####
hub_score = huber.score(X_train, y_train)
print("Huber Coeffiecient of Determination: ", hub_score)
# For some reason, ".score" method was called but nothing done with it.␣
 ↪Corrected.

#printing the coefficients and the intercept of the Huber Regression fucntion
print("Huber coefficients:", huber.coef_)
print("Huber Intercept: ", huber.intercept_)
```

```
Huber Coeffiecient of Determination:  0.4923624769253775
Huber coefficients: [ 4.81476834e-01  5.56889978e-01  8.04491248e-01
-2.73971316e-06]
Huber Intercept:  0.06461643204739986
```

Prediction

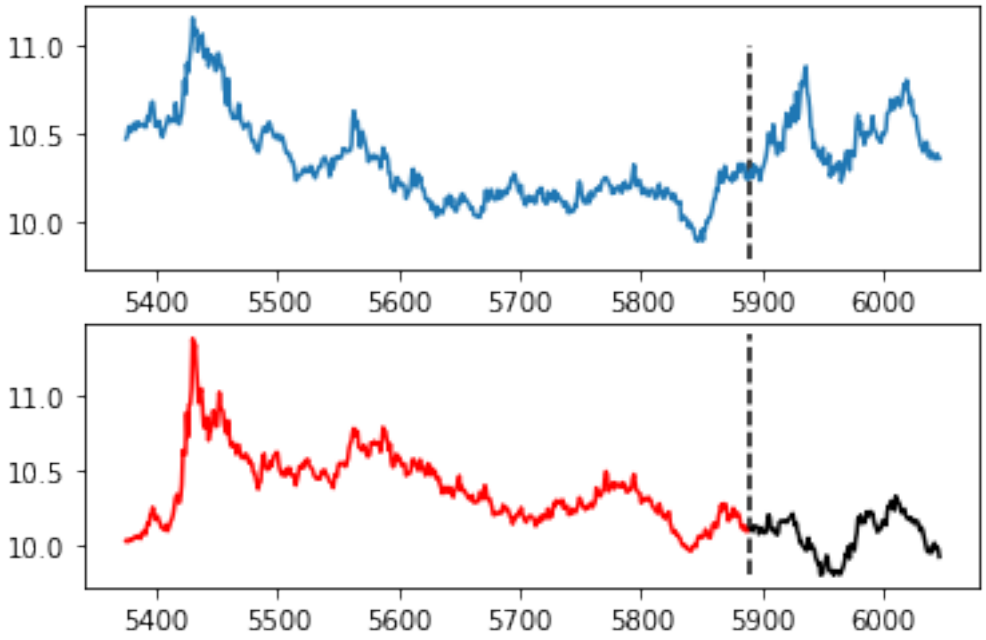
[ ]:
```
#Predicting
y_hub_pred_train = huber.predict(X_train) #Huber Regression Prediction based on␣
 ↪Training Data
y_hub_pred_val = huber.predict(X_val) #Huber Regression Prediction based on␣
 ↪Validation Data
y_hub_pred_test = huber.predict(X_test) #Huber Regression Prediction based on␣
 ↪Test Data

# Calculating Training Error
err_hub_train = HuberLoss(y_train-y_hub_pred_train,1)
err_hub_train = np.mean(err_hub_train)

#plt.scatter(y_test, y_hub_pred_test)
# PLOTTING THE TRUE LABEL & PREDICTION
plt.subplot(2,1,1)
plt.plot(tot_time[5375:5890], y_tot[5375:5890])
plt.grid()
plt.subplot(2,1,2)
plt.plot(val_time, y_hub_pred_val, 'r')
plt.grid()
```



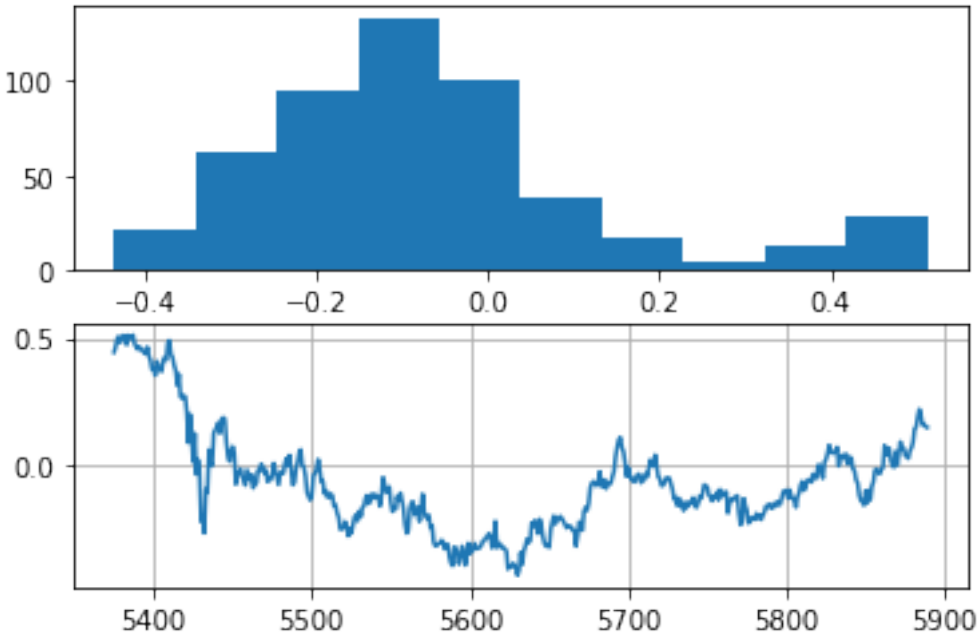
[ ]:
```
# PLOTTING THE TRUE LABEL & PREDICTION
plt.subplot(2,1,1)
plt.plot(tot_time[5375:], y_tot[5375:])
```

```
plt.vlines(5890,9.8,11, linestyles='dashed')
plt.subplot(2,1,2)
plt.plot(val_time, y_hub_pred_val, 'r')
plt.plot(test_time, y_hub_pred_test, 'k')
plt.vlines(5890,9.8,11.4, linestyles='dashed')
```

[ ]:
```
<matplotlib.collections.LineCollection at 0x7fb5d5aacd50>
```



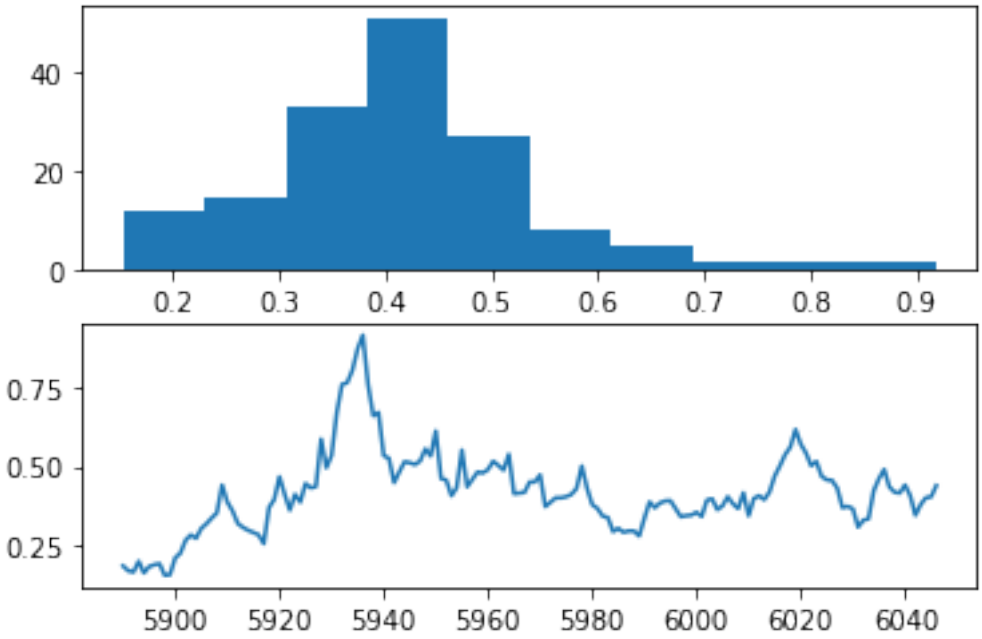
[ ]:
```
# Plotting the VALIDATION Residuals ()
plt.subplot(2,1,1)
hub_val_residuals = y_val - y_hub_pred_val
plt.hist(hub_val_residuals)
#display(np.transpose(y_hub_pred_test))
plt.subplot(2,1,2)
plt.plot(val_time, hub_val_residuals)
plt.grid()

# Calculating VALIDATION Error
err_hub_val = HuberLoss(y_val-y_hub_pred_val, 1)
err_hub_val = np.mean(err_hub_val)
```



Plotting the Residuals and Calculating Training and Validation Loss

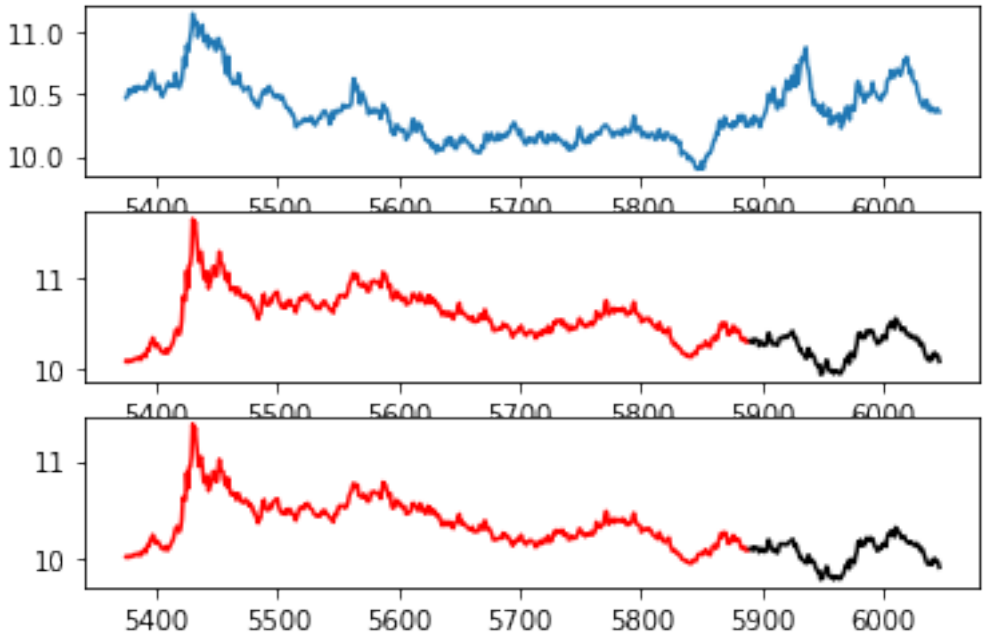
[ ]:
```
#Plotting the TEST Residuals (Histogram + Timeseries)
plt.subplot(2,1,1)
plt.hist(y_test - y_hub_pred_test)
hub_test_residuals = y_test - y_hub_pred_test
#display(np.transpose(y_hub_pred_test))
plt.subplot(2,1,2)
plt.plot(test_time, hub_test_residuals)

# Calculating TEST Error
err_hub_test = HuberLoss(y_test-y_hub_pred_test, 1)
err_hub_test = np.mean(err_hub_test)

#Printing the Errors
print("Huber Training Error: ", err_hub_train)
print("Huber Validation Error: ", err_hub_val)
print("Huber Test Error: ", err_hub_test)
```

```
Huber Training Error:  0.08742630789740509
Huber Validation Error:  0.022789866011089358
Huber Test Error:  0.09518700260390783
```



[ ]:
```
# COMPARING LINEAR AND HUBER PREDICTIONS

plt.subplot(3,1,1)
plt.plot(tot_time[5375:], y_tot[5375:])
plt.subplot(3,1,2)
plt.plot(val_time, y_pred_val, 'r')
plt.plot(test_time, y_pred_test, 'k')
plt.subplot(3,1,3)
plt.plot(val_time, y_hub_pred_val, 'r')
plt.plot(test_time, y_hub_pred_test, 'k')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7fb5d70f60d0>]
```



[ ]:
```
plt.scatter(y_pred_val, y_hub_pred_val)
# NOTE by Patrick: our models are almost the same, except for slight␣
 ↪differences.
```

[ ]: <matplotlib.collections.PathCollection at 0x7fb5d7362150>